DOMINIK HIRT @ JUG SAXONY 02.11.2023

JAVA PERSISTENZ JENSEITS VON JPA

WINCOR NIXDORF

BOSCH

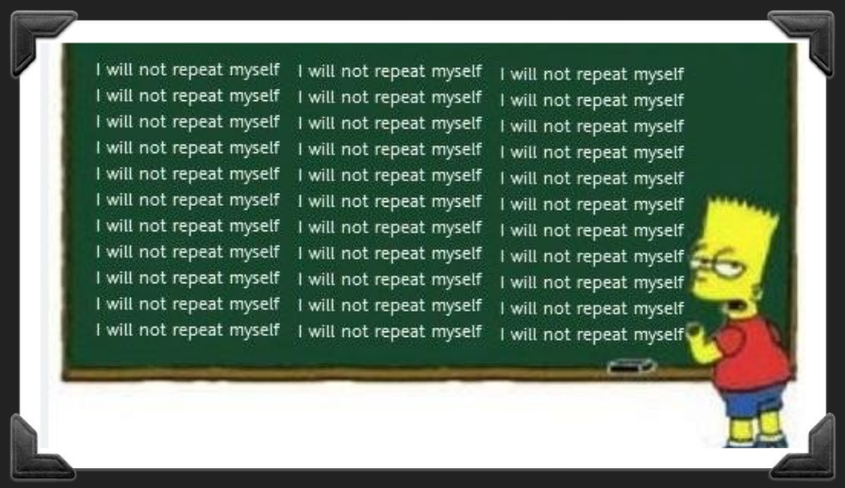
kaiser

AGENDA

- Warum kein Hibernate / JPA ?
- Welche Alternativen gibt es ?
- Warum JOOQ ?
- Wie benutze ich JOOQ ?
- Wie wirkt sich das aus ?

WARUM KEIN HIBERNATE / JPA ?

WARUM KEIN HIBERNATE / JPA ?



	Efficient	
one-to-many	@ManyToOne	@OneToMany (mappedBy=...)
one-to-one	@OneToOne @MapsId	@OneToOne + BE (mappedBy=...)
many-to-many	@ManyToMany Set<Post>	@ManyToOne @OneToMany

	Less efficient		
one-to-many	@OneToMany @JoinColumn	@OneToMany Set<Post>	@OneToMany @OrderColumn (name = ...)
one-to-one	@OneToOne (mappedBy=...)	@OneToOne + BE (mappedBy=..., optional=true)	
many-to-many	@ManyToMany @OrderColumn(name = ...) List<Post>		

	Least efficient
one-to-many	@OneToMany List<Post>
one-to-one	
many-to-many	@ManyToMany List<Post>

WARUM KEIN HIBERNATE / JPA ?

https://www.toptal.com/java/how-hibernate-ruined-my-career

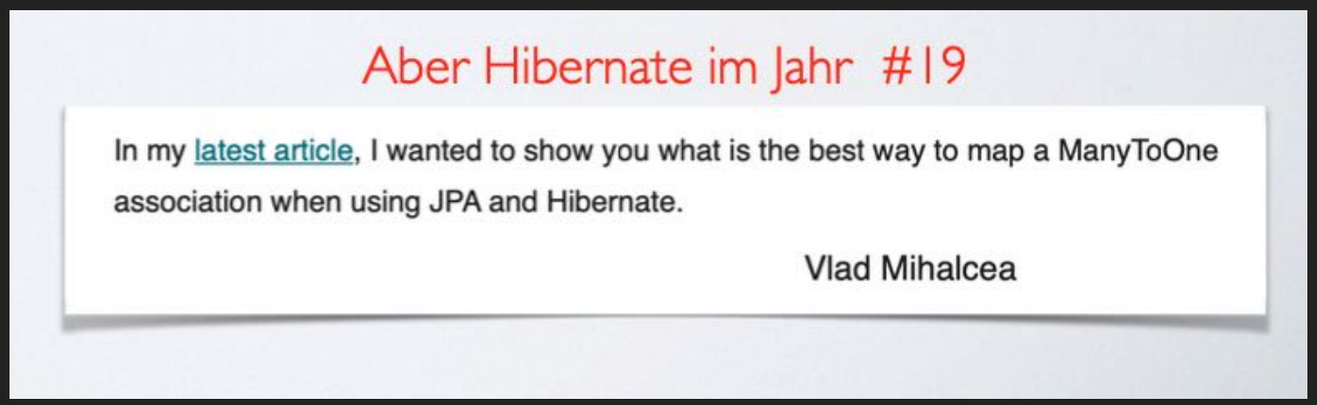
WARUM KEIN HIBERNATE / JPA ?


```java
@Entity
public class TodoEntity {

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private int id;

    private String text;

    @Column(name = "due_date")
    private LocalDate dueDate;

    private LocalDateTime notify;

    @Column(name = "modified_at")
    private LocalDate modifiedAt;

    @Column(name = "modified_by")
    private String modifiedBy;

    // constructor(s)

    // setter + getter
}
```

```sql
CREATE TABLE todo (
    id   INTEGER NOT NULL GENERATED BY DEFAULT AS IDENTITY,
    text VARCHAR NOT NULL,
    due_date DATE,
    notify TIMESTAMP,
    modified_at TIMESTAMP NOT NULL,
    modified_by VARCHAR
)
```

WARUM KEIN HIBERNATE / JPA ?

	Efficient		Less efficient			Least efficient
one-to-many	@ManyToOne	@OneToMany (mappedBy=...)	@OneToMany @JoinColumn	@OneToMany Set<Post>	@OneToMany @OrderColumn (name = ...)	@OneToMany List<Post>
one-to-one	@OneToOne @MapsId	@OneToOne + BE (mappedBy=...)	@OneToOne (mappedBy=...)	@OneToOne + BE (mappedBy=..., optional=true)		
many-to-many	@ManyToMany Set<Post>	@ManyToOne @OneToMany	@ManyToMany @OrderColumn(name = ...) List<Post>			@ManyToMany List<Post>



https://stackoverflow.com/questions/62038306/jpa-relationships-what-could-be-stand-for

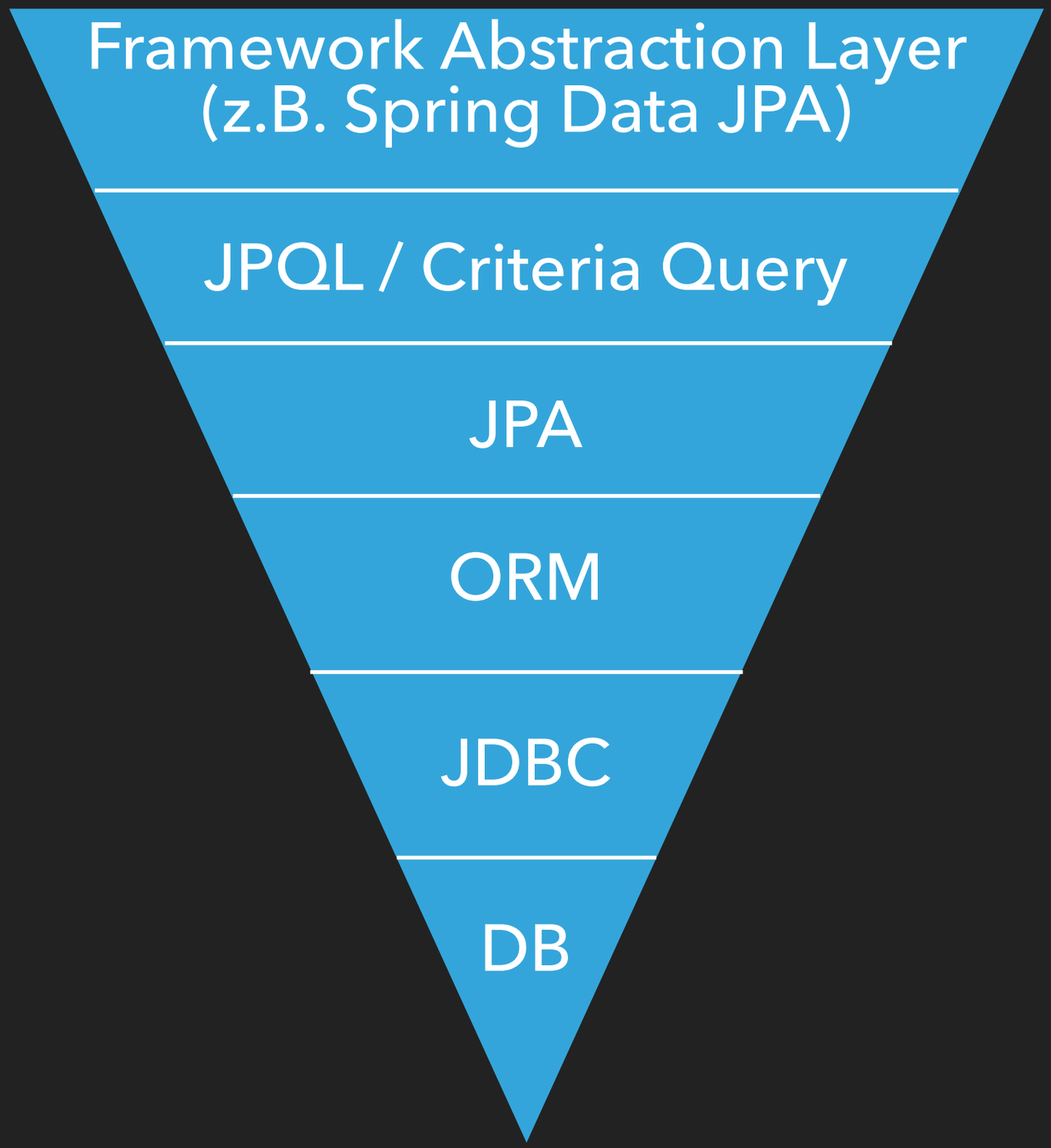
Aber das Schlimmste überhaupt ....

WARUM KEIN HIBERNATE / JPA ?

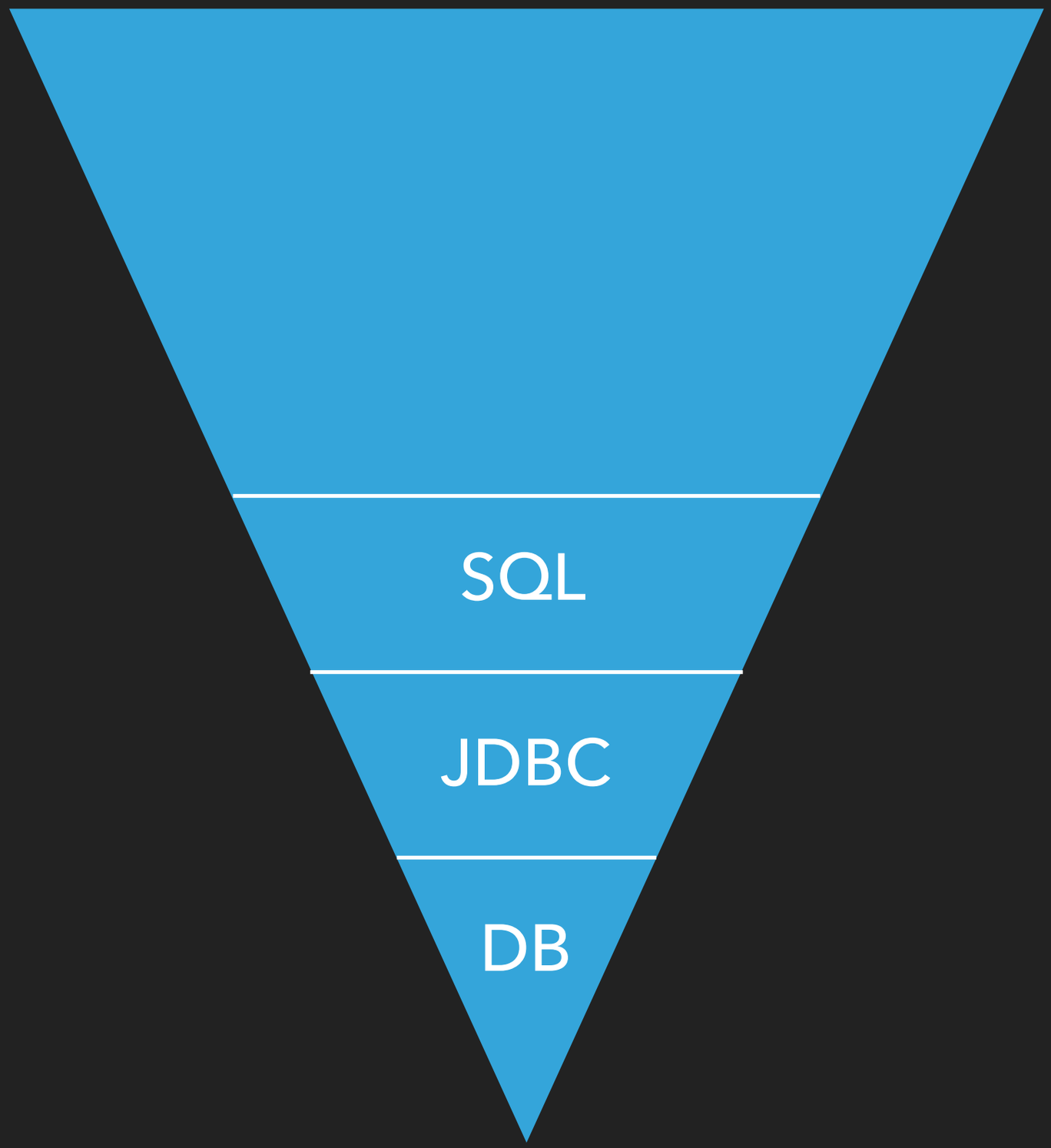
```
@Query(nativeQuery = true, value = "WITH letzter_hm AS (\n" +
        "  SELECT nadminunitno, pn_caretaker\n" +
        "    FROM (\n" +
        "        SELECT caretaker_adminunits.nadminunitno, caretaker_adminunits.pn_caretaker, ROW_NUMBER() \n" +
        "            OVER (PARTITION BY caretaker_adminunits.nadminunitno ORDER BY gueltig_ab DESC) AS rowNumber\n" +
        "            FROM caretaker_adminunits\n" +
        "            JOIN adminunit ON adminunit.nadminunitno = caretaker_adminunits.nadminunitno\n" +
        "            WHERE caretaker_adminunits.nadminunitno IN (\n" +
        "                SELECT DISTINCT caretaker_adminunits.nadminunitno \n" +
        "                    FROM caretaker_adminunits\n" +
        "            ) AND (adminunit.dendofuse is NULL or adminunit.dendofuse > CURRENT_TIMESTAMP) \n" +
        "    ) X\n" +
        "    WHERE X.rowNumber = 1\n" +
        "), \n" +
        "\n" +
        "-- ermittle alle AdminUnit deren Zuständigkeit gewechselt hat seit Datum X\n" +
        "-- und bei der ein gegebenen Hausmeister Y der zuvor Verantwortliche war\n" +
        "vorletzter_hm AS (\n" +
        "  SELECT nadminunitno, pn_caretaker \n" +
        "    FROM (\n" +
        "        SELECT caretaker_adminunits.nadminunitno, caretaker_adminunits.pn_caretaker, ROW_NUMBER() \n" +
        "            OVER (PARTITION BY caretaker_adminunits.nadminunitno ORDER BY gueltig_ab DESC) AS rowNumber\n" +
        "            FROM caretaker_adminunits\n" +
        "            JOIN adminunit ON adminunit.nadminunitno = caretaker_adminunits.nadminunitno\n" +
        "            WHERE caretaker_adminunits.nadminunitno IN (\n" +
        "                SELECT DISTINCT caretaker_adminunits.nadminunitno \n" +
        "                    FROM caretaker_adminunits\n" +
        "                    WHERE caretaker_adminunits.gueltig_ab > ? \n" +
        "            ) AND (adminunit.dendofuse is NULL or adminunit.dendofuse > CURRENT_TIMESTAMP) \n" +
        "    ) Y\n" +
        "    WHERE Y.rowNumber = 2\n" +
        ") \n" +
        "select nadminunitno from letzter_hm where pn_caretaker = ? \n" +
        "union \n" +
        "select nadminunitno from vorletzter_hm where pn_caretaker = ? ;\n" +
        "")
public List<Integer> findByCaretaker(Date startDate, String pnCaretaker1, String pnCaretaker2);
```

#epic fail

JAVA PERSISTENT PYRAMIDE

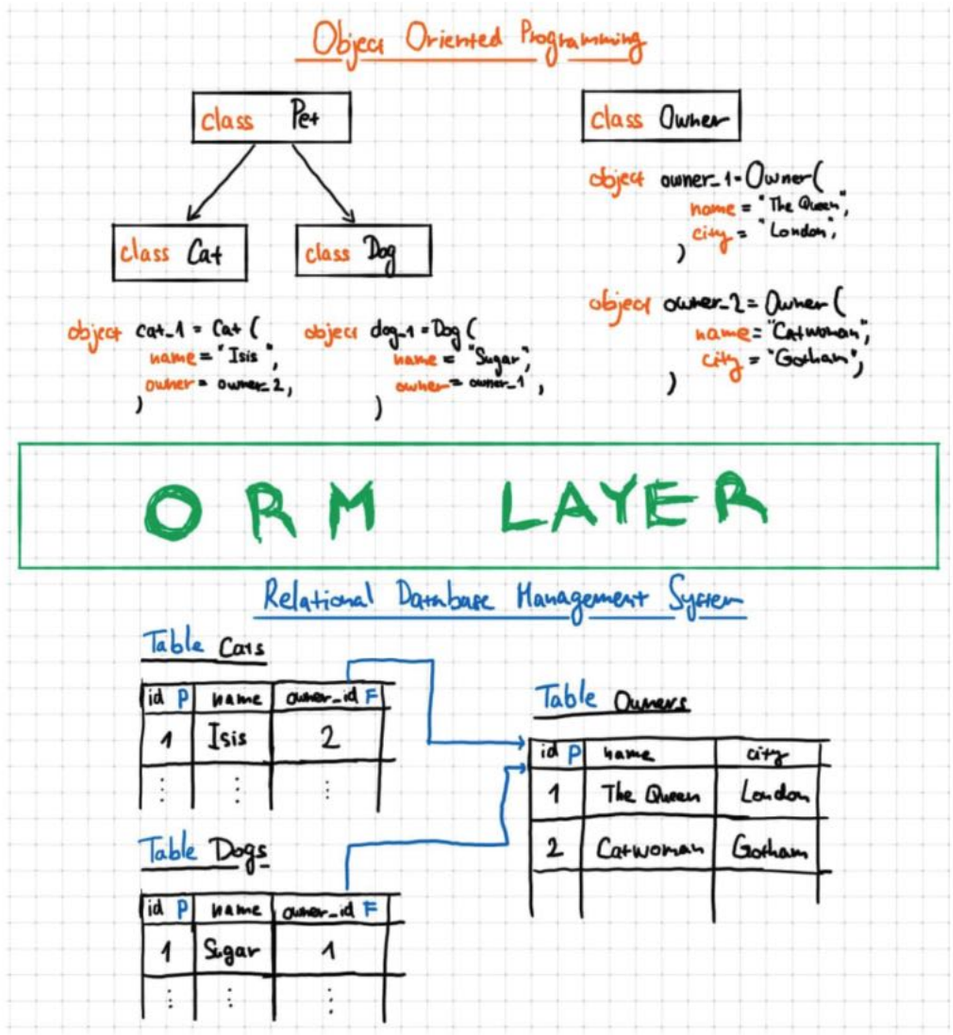


JAVA PERSISTENT PYRAMIDE



WARUM KEIN HIBERNATE / JPA ?

Impedance Mismatch



[https://dev.to/tinazhouhui]

VORURTEILE, VORURTEILE *

- SQL ist schwer zu lernen
- SQL ist veraltet
- SQL ist nicht performant
- SQL ist unflexibel
- SQL ist nur für Datenbankadministratoren
- SQL ist nicht sicher
- SQL zu schreiben ist zeitaufwändig

* ChatGPT

WAS IST SQL WIRKLICH ?

- SQL was never meant to be abstracted.
- SQL was never meant to be object-oriented.
- SQL was never meant to be anything other than …
  SQL!

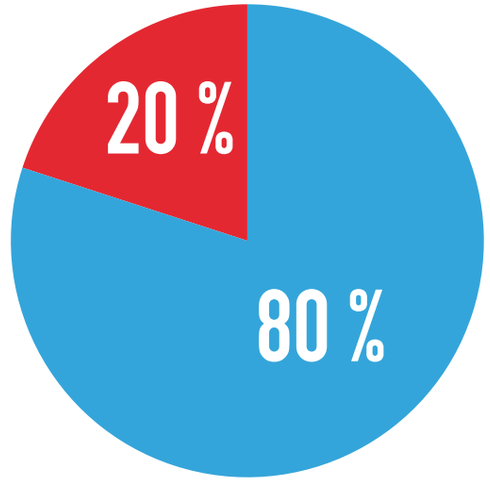
IMPERATIV

Wie soll etwas
ausgeführt werden

DEKLARATIV

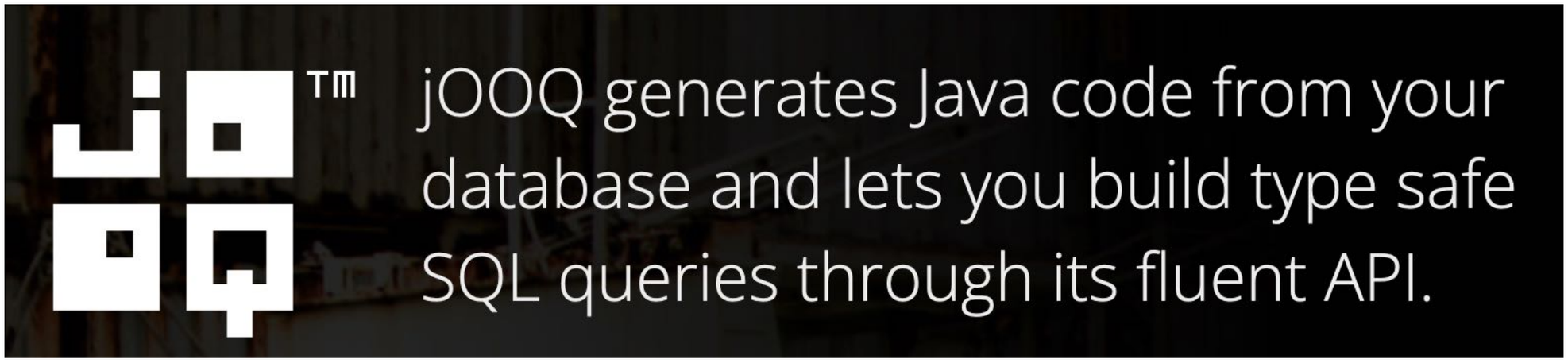
Was soll
herauskommen

ERP / BUSINESS SOFTWARE

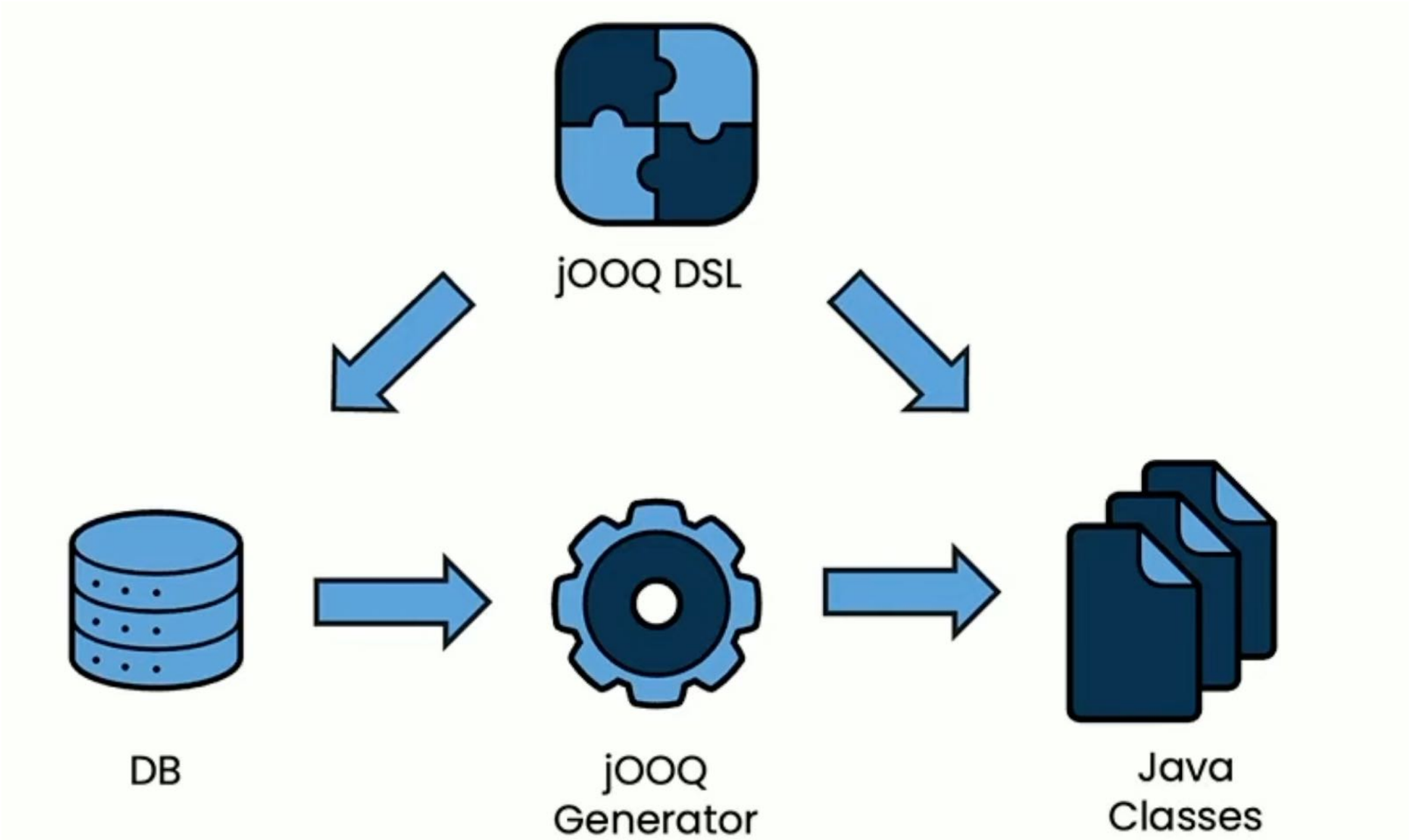


- Datenmodell existiert viel länger als Anwendungen
- Änderungen am Datenmodell durch Hibernate 🙈
- Nicht das Datenmodell ist hierarchisch sondern (einige) Darstellungsweisen

http://www.jooq.org


jOOQ generates Java code from your
database and lets you build type safe
SQL queries through its fluent API.

ARCHITEKTUR



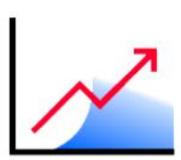
CODE GENERATION

https://www.jooq.org/doc/3.18/manual/code-generation/codegen-configuration

LIQUIBASE – DATABASE MIGRATION TOOLS

Flexible database change

Easily define changes in SQL, XML, JSON, or YAML.



Version control for your database

Order changes and standardize development.

Built for developers

Control when, where, and how database changes are deployed.

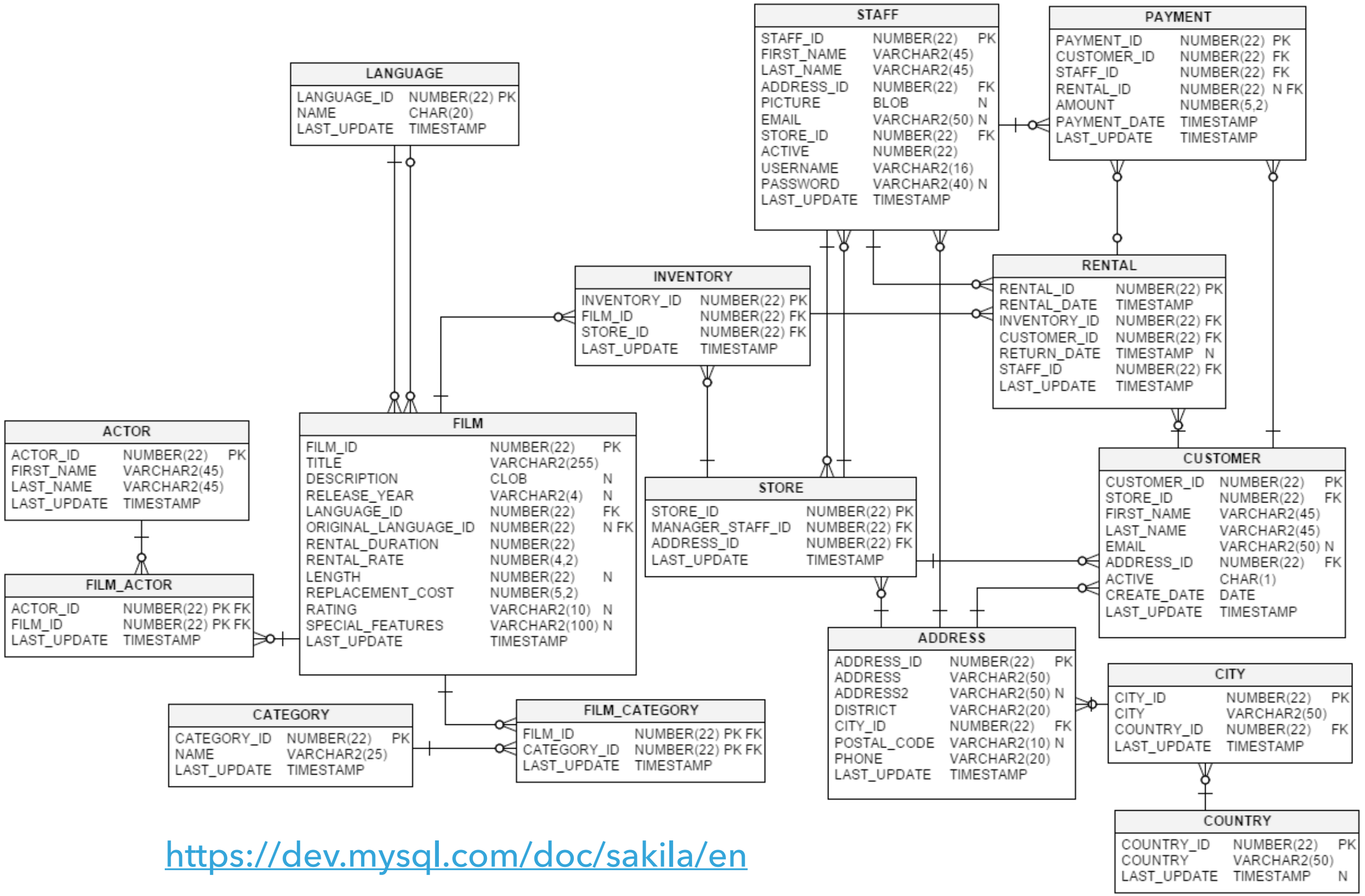
CODE GENERATION

- active record pattern
- DAO
- many more ...

```xml
<plugin>
    <groupId>org.jooq</groupId>
    <artifactId>jooq-codegen-maven</artifactId>
    <executions>
        <execution>
            <id>default</id>
            <phase>generate-sources</phase>
            <goals>
                <goal>generate</goal>
            </goals>
            <configuration>
                <jdbc>
                    <driver>org.postgresql.Driver</driver>
                    <url>${database}</url>
                    <user>${username}</user>
                    <password>${password}</password>
                </jdbc>
                <generator>
                    <database>
                        <inputSchema>public</inputSchema>
                        <excludes>Databasechangelog.*</excludes>
                        <recordTimestampFields>modified_at</recordTimestampFields>
                    </database>
                    <generate>
                        <pojos>true</pojos>
                        <pojosEqualsAndHashCode>true</pojosEqualsAndHashCode>
                    </generate>
                    <target>
                        <packageName>de.hub28.demo.generated</packageName>
                        <directory>target/generated-sources/jooq</directory>
                    </target>
                </generator>
            </configuration>
        </execution>
    </executions>
    <dependencies>
        <dependency>
            <groupId>org.postgresql</groupId>
            <artifactId>postgresql</artifactId>
        </dependency>
    </dependencies>
</plugin>
```

https://www.jooq.org/doc/3.18/manual/code-generation/codegen-configuration

<DEMO/>



https://dev.mysql.com/doc/sakila/en

TYPE SAFE FETCH

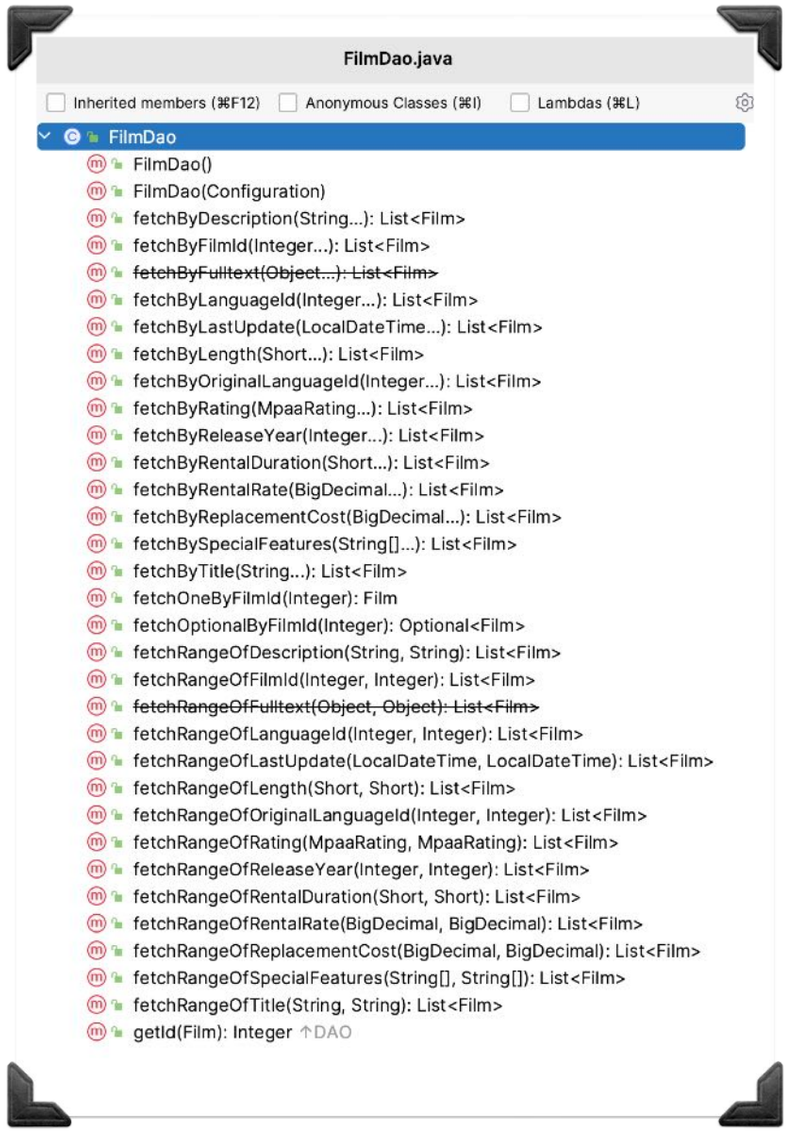
```java
// The "standard" fetch
Result<R> fetch();

// The "standard" fetch when you know your query returns at most one record. This may return null.
R fetchOne();

// The "standard" fetch when you know your query returns exactly one record. This never returns null.
R fetchSingle();

// The "standard" fetch when you know your query returns at most one record.
Optional<R> fetchOptional();

// The "standard" fetch when you only want to fetch the first record
R fetchAny();
```

https://www.jooq.org/doc/current/manual/sql-execution/fetching

TYPE SAFE FETCH

```
// The "create" reference is an instance of DSLContext

// Fetching only book titles (the two calls are equivalent):
List<String> titles1 = create.select().from(BOOK).fetch().getValues(BOOK.TITLE);
List<String> titles2 = create.select().from(BOOK).fetch(BOOK.TITLE);
String[]     titles3 = create.select().from(BOOK).fetchArray(BOOK.TITLE);

// Fetching only book IDs, converted to Long
List<Long> ids1 = create.select().from(BOOK).fetch().getValues(BOOK.ID, Long.class);
List<Long> ids2 = create.select().from(BOOK).fetch(BOOK.ID, Long.class);
Long[]     ids3 = create.select().from(BOOK).fetchArray(BOOK.ID, Long.class);

// Fetching book IDs and mapping each ID to their records or titles
Map<Integer, BookRecord> map1 = create.selectFrom(BOOK).fetch().intoMap(BOOK.ID);
Map<Integer, BookRecord> map2 = create.selectFrom(BOOK).fetchMap(BOOK.ID);
Map<Integer, String>     map3 = create.selectFrom(BOOK).fetch().intoMap(BOOK.ID, BOOK.TITLE);
Map<Integer, String>     map4 = create.selectFrom(BOOK).fetchMap(BOOK.ID, BOOK.TITLE);

// Group by AUTHOR_ID and list all books written by any author:
Map<Integer, Result<BookRecord>> group1 = create.selectFrom(BOOK).fetch().intoGroups(BOOK.AUTHOR_ID);
Map<Integer, Result<BookRecord>> group2 = create.selectFrom(BOOK).fetchGroups(BOOK.AUTHOR_ID);
Map<Integer, List<String>>       group3 = create.selectFrom(BOOK).fetch().intoGroups(BOOK.AUTHOR_ID, BOOK.TITLE);
Map<Integer, List<String>>       group4 = create.selectFrom(BOOK).fetchGroups(BOOK.AUTHOR_ID, BOOK.TITLE);
```

DAO – DATA ACCESS OBJECTS / REPOSITORY



MULTISET (SUPERPOWER 💪)

```
public List<StahlbauAufmassPlanungSystem> fetchNested(Integer projectId) {
    return jooq.select(

        // Hierarchie Ebene 1
        PLANUNG_SYSTEM.ID,
        PLANUNG_SYSTEM.NAME,

        // Hierarchie Ebene 2 - liste der planungen
        DSL.multiset(DSL.select(
            PLANUNG.ID,
            DOCUMENT.FILENAME,

            // Hierarchie Ebene 3 - liste der typicals
            DSL.multiset(DSL.select(
                STAHLBAU_GRUPPE.ID,
                STAHLBAU_GRUPPE.NAME,
                STAHLBAU_GRUPPE.MENGE,

                // Hierarchie Ebene 4 - liste der bauteile eines Typical
                DSL.multiset(DSL.select(
                            PLANUNG_BAUTEIL.ID,
                            PLANUNG_BAUTEIL.NAME,
                            PLANUNG_BAUTEIL.RAFF_MENGE,
                            PLANUNG_BAUTEIL.RAFF_EINHEIT
                        )
                        .from(PLANUNG_BAUTEIL)
                        .where(PLANUNG_BAUTEIL.STAHLBAU_TYPICAL_ID.eq(STAHLBAU_GRUPPE.ID))
                        .orderBy(PLANUNG_BAUTEIL.NAME)
                ).convertFrom(r -> r.map(Records.mapping(StahlbauAufmassBauteil::new))))

                .from(STAHLBAU_GRUPPE)
                .join(PLANUNG_BAUTEIL)
                    .on(PLANUNG_BAUTEIL.STAHLBAU_TYPICAL_ID.eq(STAHLBAU_GRUPPE.ID)
                        .and(PLANUNG_BAUTEIL.PROJECT_ID.eq(projectId)))
                .where(PLANUNG_BAUTEIL.PLANUNG_ID.eq(PLANUNG.ID))
                .groupBy(STAHLBAU_GRUPPE.NAME, STAHLBAU_GRUPPE.ID)
                .orderBy(STAHLBAU_GRUPPE.NAME)
            ).convertFrom(r -> r.map(Records.mapping(StahlbauAufmassTypical::new))))

            .from(PLANUNG)
            .join(DOCUMENT).onKey()
            .where(PLANUNG.SYSTEM_ID.eq(PLANUNG_SYSTEM.ID)) // join mit äußerer query
            .orderBy(DOCUMENT.FILENAME)
        ).convertFrom(r -> r.map(Records.mapping(StahlbauAufmassPlanung::new))))

        .from(PLANUNG_SYSTEM)
        .where(PLANUNG_SYSTEM.PROJECT_ID.eq(projectId)
            .and(PLANUNG_SYSTEM.NAME.containsIgnoreCase(Constants.PLANUNG_SYSTEM_STAHLBAU_PREFIX)))
        .fetch(Records.mapping(StahlbauAufmassPlanungSystem::new));
}
```

<DEMO/>

CRUD

```java
// Refresh a record from the database.
void refresh() throws DataAccessException;

// Store (insert or update) a record to the database.
int store() throws DataAccessException;

// Delete a record from the database
int delete() throws DataAccessException;
```

https://www.jooq.org/doc/3.18/manual/sql-execution/crud-with-updatablerecords/simple-crud/

TYPE SAFETY

RUNTIME

Strings

COMPILE

Typen / API / DSL

TYPE SAFETY

```
jooq.select(FILM.TITLE, FILM.INFO)
           .from(FILM)
           .where(FILM.TITLE.startsWith("B"))
           .orderBy(FILM.TITLE.asc())
           .fetch();
```

Compile error

IDE auto completion

```
jooq.select(FILM.TITLE, FILM.DESCRIPTION)
           .from(FILM)
           .where(FILM.TITLE.startsWith("B"))
           .orderBy(FILM.TITLE.asc())
           .fetch();
```

<DEMO/>

OPTIMISTIC LOCKING

```xml
<plugin>
   <groupId>org.jooq</groupId>
   <artifactId>jooq-codegen-maven</artifactId>
   <executions>
       <execution>
           <phase>generate-sources</phase>
           <goals>
               <goal>generate</goal>
           </goals>
           <configuration>
               <generator>
                   <database>
                      ....
                      <recordTimestampFields>last_update</recordTimestampFields>
                      ....
                   </database>
             </generator>
          </configuration>
      </execution>
   </executions>
</plugin>
```

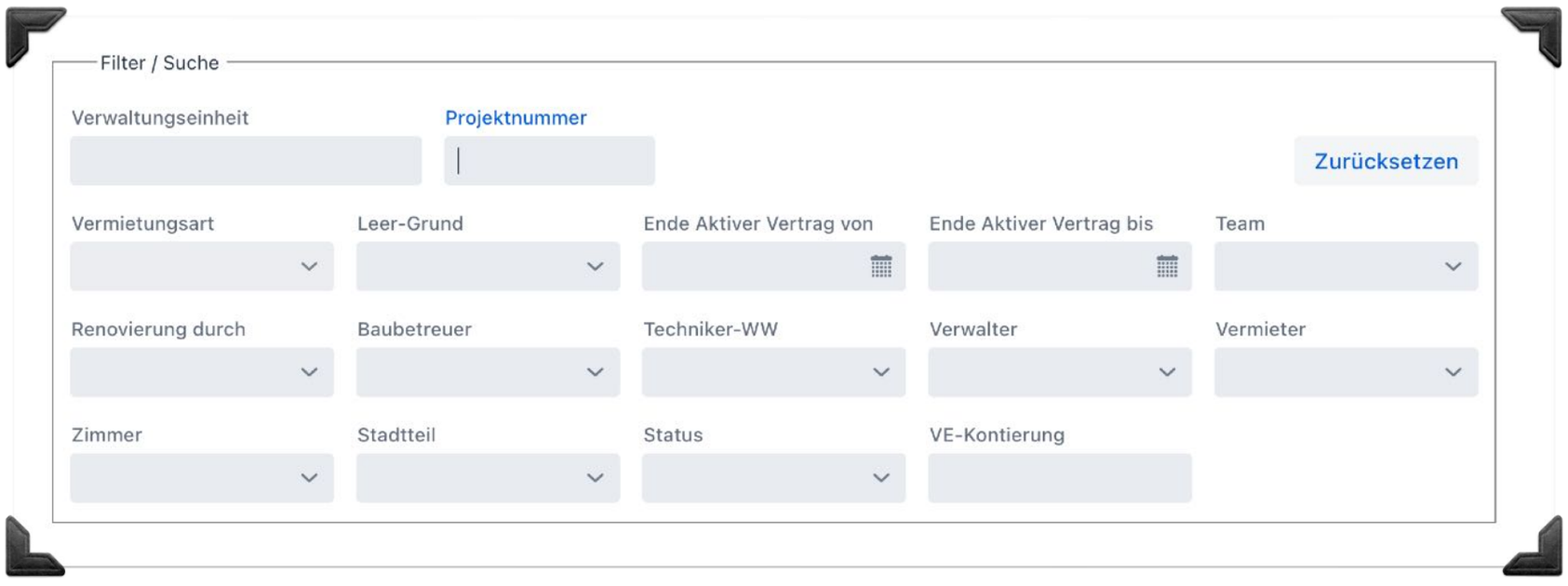
```java
@Bean
public DefaultConfigurationCustomizer configurationCustomizer() {
    return (DefaultConfiguration defaultConfig) -> {
        defaultConfig.settings()
            // ...
            .withUpdateRecordTimestamp(true)
            .withExecuteWithOptimisticLocking(true)
            // ..
         ;
        defaultConfig.set(new JooqExceptionTranslator());
    };
}
```

https://www.jooq.org/doc/latest/manual/sql-execution/crud-with-updatablerecords/optimistic-locking/

<DEMO/>

DYNAMIC QUERIES

Such- und Auskunfsdialoge



DYNAMIC QUERIES

```
Condition whereClause = DSL.noCondition();

if (request.getProjectType() != null) {
   whereClause = whereClause.and(PROJECT.TYPE.eq(request.getProjectType().name()));
} else {
    List<ProjectType> types = Arrays.asList(ProjectType.values());
    whereClause = whereClause.and(PROJECT.TYPE.in(types));
}

if (request.getCompany() != null) {
    whereClause = whereClause.and(PROJECT.COMPANY.eq(request.getCompany().getId()));
}

if (request.getSales() != null) {
    whereClause = whereClause.and(PROJECT.SALES.eq(request.getSales().getId()));
}

if (request.getSeason() != null) {
    whereClause = whereClause.and(PROJECT.SEASON.eq(request.getSeason().getId()));
}

if (request.getCustomer() != null) {
    whereClause = whereClause.and(PROJECT.CUSTOMER.eq(request.getCustomer().getId()));
}

if (request.getFabricationType() != null) {
    whereClause = whereClause.and(PROJECT.FABRICATIONTYPE.eq(request.getFabricationType().name()));
}

List<ProjectOverviewResponseDTO> result = statement.where(whereClause)
        .groupBy(PROJECT_POSITIONS.KUNDENREFERENZ)
        .orderBy(PROJECT.PROJECTDATE.desc())
        .fetchInto(ProjectOverviewResponseDTO.class);
```

👉 kann aber generalisiert werden

DYNAMIC QUERIES

```
var selectJoinStep = this.jooq.select(PLANUNG_BAUTEIL.ID,
    PLANUNG_BAUTEIL.NAME,
    PLANUNG_BAUTEIL.TYPE,
    DSL.multiset(
        DSL.select(PLANUNG_BAUTEIL_EIGENSCHAFT.planungBauteilAttribute().NAME,
                   PLANUNG_BAUTEIL_EIGENSCHAFT.VALUE)
        .from(PLANUNG_BAUTEIL_EIGENSCHAFT)
        .where(PLANUNG_BAUTEIL_EIGENSCHAFT.BAUTEIL.eq(PLANUNG_BAUTEIL.ID)))
        .as(PlanungBauteilDto.MEMBER_NAME_EIGENSCHAFTEN)
        .convertFrom(r -> r.map(Records.mapping(PlanungBauteilDtoEigenschaft::new))),
    PLANUNG_BAUTEIL.BESCHAFFUNG_STATUS,
    DOCUMENT.FILENAME,
    PLANUNG_SYSTEM.NAME,
    PLANUNG_BAUTEIL.STAHLBAU_TYPICAL, PLANUNG_BAUTEIL.STAHLBAU_TYPICAL_ID
    )
    .from(PLANUNG_BAUTEIL)
    .join(PLANUNG).onKey()
    .join(DOCUMENT).onKey()
    .leftJoin(PLANUNG_SYSTEM).on(PLANUNG_SYSTEM.ID.eq(PLANUNG.SYSTEM_ID));

if (CollectionUtils.isNotEmpty(filter.getPlanungSystemIds())) {
    whereClause = whereClause.and(PLANUNG_SYSTEM.ID.in(filter.getPlanungSystemIds()));
}

if (this.attributeMedium == null) {
    this.attributeMedium = this.jooq.select(PLANUNG_BAUTEIL_ATTRIBUTE.ID)
        .from(PLANUNG_BAUTEIL_ATTRIBUTE)
        .where(PLANUNG_BAUTEIL_ATTRIBUTE.NAME.eq("Medium"))
        .fetchOneInto(Integer.class);
}

if (CollectionUtils.isNotEmpty(filter.getPlanungMedium())) {
    selectJoinStep = selectJoinStep
        .join(PLANUNG_BAUTEIL_EIGENSCHAFT).onKey();
    whereClause = whereClause.and(PLANUNG_BAUTEIL_EIGENSCHAFT.ATTRIBUTE.eq(this.attributeMedium)
        .and(PLANUNG_BAUTEIL_EIGENSCHAFT.VALUE.in(filter.getPlanungMedium())));
}

List<ZuordnungPlanungBauteilDto> bauteilDtos = selectJoinStep
    .where(whereClause)
    .orderBy(PLANUNG_BAUTEIL.NAME)
    .fetch(Records.mapping(ZuordnungPlanungBauteilDto::new));
```

RECORD/EXECUTION LISTENER

WEITERE VORTEILE

- JOOQ skaliert besser als Hibernate (Ap Start: Hibernate scannt ale entities)
- forced types (am Beispiel ZE-TIM Money)
- Optimistic locking

TIME

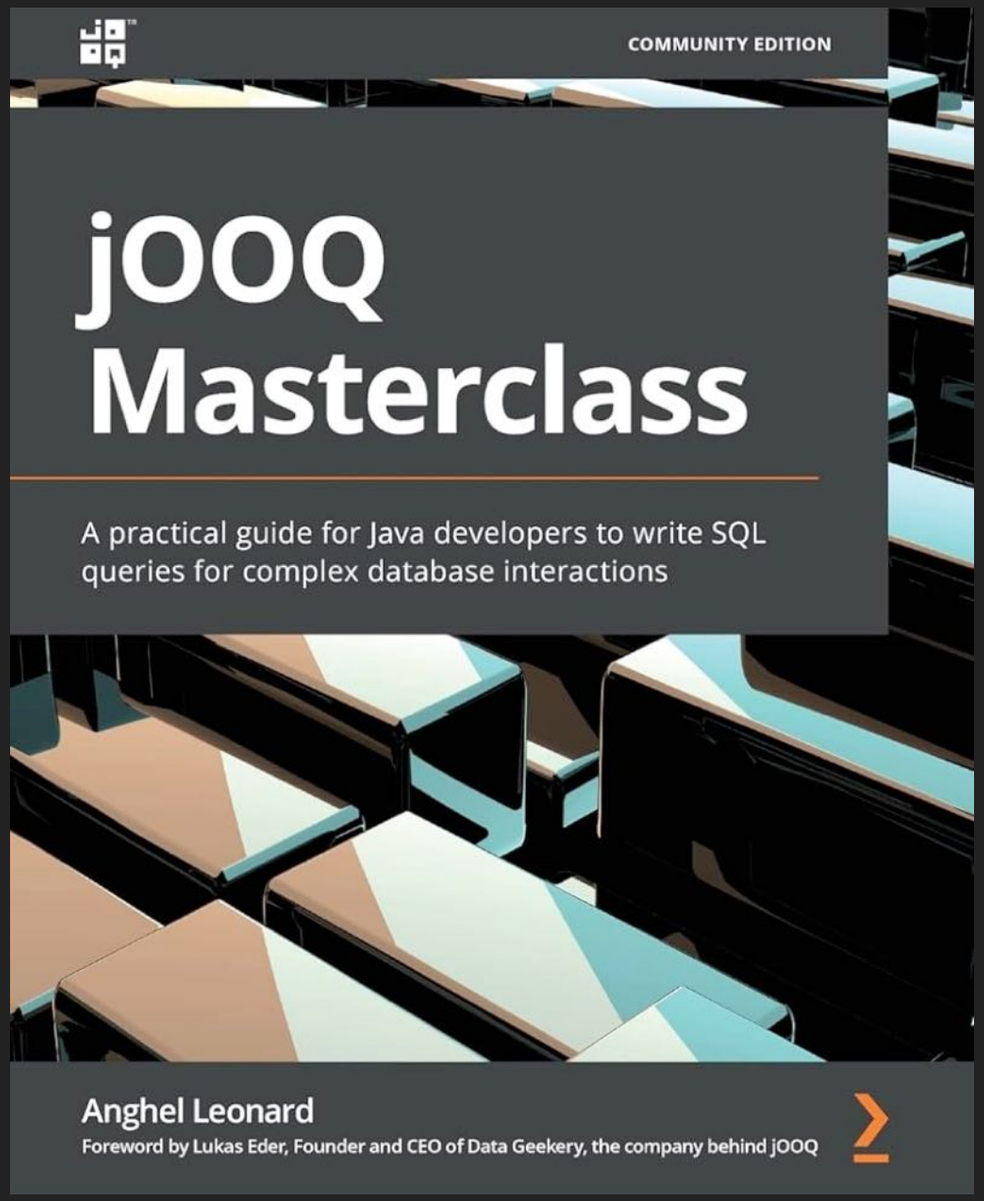
FOR

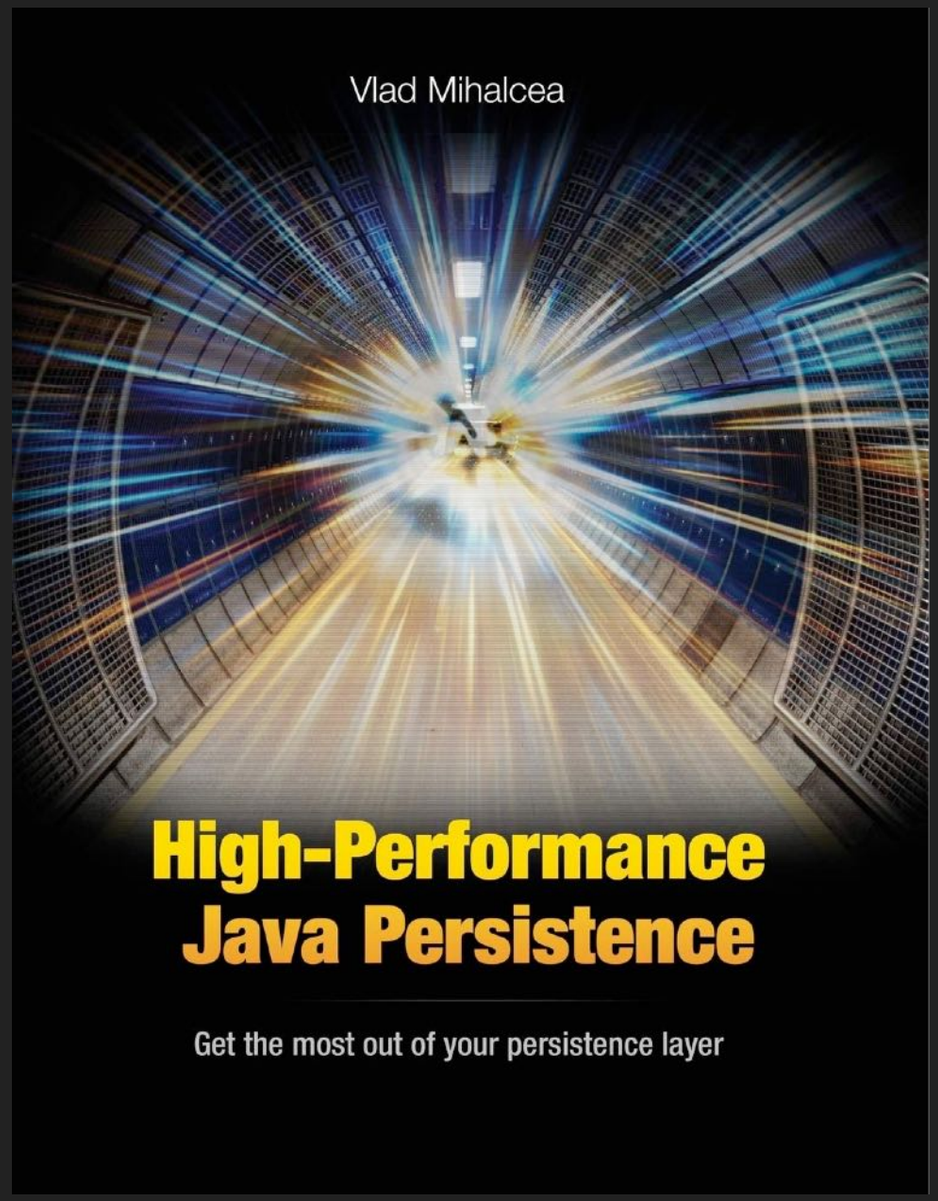
RECAP

USING JOOQ WILL HELP LEARN AND UNDERSTAND SQL

Lukas Eder

RECAP


COMMUNITY EDITION
jOOQ Masterclass
A practical guide for Java developers to write SQL queries for complex database interactions
Anghel Leonard
Foreword by Lukas Eder, Founder and CEO of Data Geekery, the company behind jOOQ


Vlad Mihalcea
High-Performance Java Persistence
Get the most out of your persistence layer

JPA VS. JOOQ

- Nicht technisch
- Gedanklich
  - entity state transitions vs. data set transformations
- object first vs. database first

Choose your jOOQ Edition

Choose your jOOQ Edition depending on the types of databases that you want to support.

Open Source

Use this free edition with your favourite *Open Source DB* using the popular Apache Software License 2.0!

jOOQ 3.18.7
for unlimited use

free

Express

You're a small startup or an individual, working with *Oracle Express, SQL Server Express, and/or MS Access*?

jOOQ 3.18.7 free trial
for 30 days

99 € excl. VAT

per floating developer workstation and year
including ongoing maintenance and support

Buy Now »

Professional

Popular Choice

You're a small or medium-sized company wanting to work with *Oracle, SQL Server, and/or MS Access* and you're looking for basic support?

jOOQ 3.18.7 free trial
for 30 days

399 € excl. VAT

per floating developer workstation and year
including ongoing maintenance and support

Buy Now »

Enterprise

You're a large company working with many types of *enterprise databases* and you're looking for premium support?

jOOQ 3.18.7 free trial
for 30 days

799 € excl. VAT

per floating developer workstation and year
including ongoing maintenance and support

Buy Now »

<DANKE/>

dominik.hirt@hub28.de

@todo42